● ASSIGNMENT - 2: DTB : SUPPLY CHAIN MANAGEMENT

Team Members:
Kshitij Gutpa: 2021201075
Annapoorani Anantharaman: 2021201017

Project structure:

```
src
├── components
          ├── AdminComp
                    ├──AddManufacturer.js
                    ├──AddSupplier.js
          ├──ManuComp
                    ├──Bid.js
                    ├──Price.js
                    ├──Product.js
          ├──Admin.js
          ├──Buyer.js
          ├──Manufacturer.js
          ├──Supplier.js
├── contracts
          ├──SupplyChain.sol
├── migrations
          ├──1_SupplyChain_migration.sol
├──tests
          ├──SupplyChain_test.sol
├── Main.js
├── …
```

Steps to run:

- Unzip the zip file
- Install the following modules :
  -ganache
  -npm
  -metamask
  -truffle
- Install the following dependencies:
  npm install
  npm install antd

Our SmartContract
Actors:

1. Supplier: There are 3 suppliers (Vedanta, MRF and CEAT). MRF and CEAT manufacture tyres. Vedanta supplies body parts.
2. Manufacturers: There are 2 clients (Tata and Maruti) which procure wheels and tyres
3. Customers/Buyers: Buy products from manufacturers.

Technology used:
Truffle, React JS, Metamask, Ganache, node js, Solidity, web3.0 framework

Functionalities and logic from UI :

- Click on “Enter” in order to start the application. The address of the user is obtained and classified into Roles(Supplier, Manufacturer, Buyer or Admin)
- Admin gets to add supplier and add manufacturer information. This is passed to our smart contract functions.
- Global objects are used as useStates in order to store objects returned by the various components.
- Supplier can set a limit using Supplier.js which is updated to smart contract. A current limit specifies his current set limit. The parent is Supplier.js
- Manufacturer can set Bid, Price and make products. The components being used are Bid.js, Price.js and Product.js - the parent component is Manufacturer.js
- Buyer can buy product by entering supplier name and selecting buy. The Buyer.js component does the buying UI, and the smart contract functionalities are called.
- The smart contract is initialised by using the web3 interface.
- The buyer can thus purchase and verify the product by testing its uid and if the parts are valid, the goals are satisfied.
- Metamask and ganache are linked to have 10 local accounts and the transactions are called.
- The header contains tabs to display supply limits and manufacturers and hide them.
- The footer contains a link to go back to the initial page.

FLOW:

- First, the init() function is called on click of enter button. The control goes to this function which sets the contract object and accounts and balance are displayed on top, using web3.js
- Initially, the role of the address is identified by passing account address to solidity and getting the type in numbers (case 0 = Admin, case 1 = Supplier, case 2 = Manufacturer, case 3 = Buyer)
- These are put in a switch case statement. The boolean variables are set their respective roles by useStates.
- The values are rendered in the return function and the corresponding components are called and values are passed as props. If admin is true, Admin.js component is called. Similarly, the components Supplier.js,Manufacturer.js and Buyer.js are called.

- A useEffect is used which checks if the dependencies change. It checks flag variables which are set by the respective components. They get set to 1 if the user clicks submit in the pages and then we reset them back to 0. If they do, it renders the values by calling functions fun0(), fun1(), fun2() and fun3() for each of the if conditions in the useEffect. These functions link to the actual functions of the smart contract.
- The smart contract then performs the necessary computations and connects to the metamask wallet. The ethers are transferred if required and the goals are achieved.

Goals:

Goal 1: Bidding is kept secret as no one can access the bids. It is internal only to the smart contract. Only the internal function to contract can open the values and check prices and quantity.
This is abstracted even from manufacturers and suppliers. This is consistent with UI, as only a particular address can set a bid of their choice. These addresses are set using metamask.
Goal 2: Optimal Supply of material is performed by the resource allocation algorithm we have defined in our smart contract function. The linking is done and it is called thus to establish goal 2. Thus it is a decentralised and trustless
supply-chain.
Goal 3: Validity of product is ensured by using unique uids for products and the individual materials. They are only incremental in nature. They cannot be decremented. The blockchain encrypts the smart contract and uses hash pointers at each stage to hash our values, so no supplication or manipulation can occur. For the same reason, double spending cannot happen as buyers can perform verification by checking the uids and pids and matching with the address of the supplier/manufacturer which we store in our mapping. These addresses are unique and cannot be duplicated. These are intrinsic properties of the blockchain, as a smart contract may use currencies as non-fungible tokens but it in itself remains unique because of the hashing properties and the way the blockchain is built. The front end reflects these changes.